34 p.

Technical Report No. 32-546

A Set of Algorithms for a Diagnosable Arithmetic Unit

Algirdas Avizienis

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

March 1, 1964

(NASA CR-53575;
TR-
(JPL- ~~Technical Report No.~~ 32-546) OTS: $3.60 ph, $1.22 mf

A Set of Algorithms for a Diagnosable Arithmetic Unit

Algirdas Avizienis

C73

Henry A. Curtis

Henry A. Curtis, *Chief*
Flight Computers and Sequencers Section

(NASA Contract NAS7-100)

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

March 1 1964

Prepared Under Contract No. NAS 7-100
National Aeronautics & Space Administration

CONTENTS

TABLES

FIGURES

ABSTRACT

18452 A

The design of a diagnosable arithmetic unit for a digital computer is discussed in this Report. Arithmetical errors are classified, and the effects of a circuit failure on an arithmetical result are considered. The methods of hardware (wired-in) checking of arithmetic are reviewed, and the cost of checking is proposed as a merit criterion. A low-cost checking algorithm for product-coded numbers is presented, and its effectiveness is evaluated with respect to various types of errors. A complete set of arithmetical algorithms for product-coded numbers is also presented. The algorithms are especially economical when the product code is chosen for the proposed low-cost checking algorithm.

Author

I. COMPUTATIONAL ERRORS IN DIGITAL ARITHMETIC

A. Introduction

Functionally, the arithmetic unit of a digital computer may be described as a "black box" which receives two types of inputs:

1. The operation code (OC), which uniquely specifies one algorithm from the several other arithmetic algorithms available in the given computer.
2. One operand (X) or a pair of operands (X, Y), which are digital numbers taken from a finite set of representable digital numbers of the given computer.

Corresponding to every **input pair** (OC, X) or **input triplet** (OC, X, Y), the black box produces, after a time interval t(OC), one of two types of outputs:

1. One **result** (S) or a pair of results (S, T) which are members of the representable set of digital numbers.
2. A **special signal** (SI) from a set of special signals indicating **singularities**. A singularity is a result which is not contained in the set of representable digital numbers. The associated **pseudo-result** may also be specified for any SI.

The set of operation codes {OC} and the set of representable digital numbers {X} to be used depends on the class of problems to which the computer is applied. The logic design of the black box arithmetic unit depends on the required operation speeds and the allowable hardware cost in terms of logic and storage circuits. The logic for arithmetic control has been included in the arithmetic unit by the above definitions. The least time (a fixed Δt) and the most hardware will be required if arithmetic is performed as table look-up of results (S, T) for every possible set of inputs (OC, X, Y). Hardware requirements are reduced and operation times increased when the results (S, T) are generated by means of arithmetical algorithms. These algorithms are exact rules for computing the results by means of a sequence of gating operations between storage circuits through combinational logic circuitry. Repeated use of the same circuits reduces hardware requirements but increases t(OC) and the complexity of control sequences for the algorithms.

B. Computational Errors

Regardless of its internal logic design, every arithmetic unit is intended to be an entirely deterministic black box, with only one result corresponding to each set of inputs. However, a failure or a temporary malfunction of a circuit in the arithmetic unit may cause a different result

(S^*) to be generated instead of the specified result (S). There are two broad categories of this event

1. The wrong operation is executed (OC^* instead of OC) because of a **control error.**
2. The wrong result is generated (S^* instead of S) because of an **arithmetical error.**

The detection of both types of computational errors is a significant problem in present-day digital computers. The more common of two possible detection approaches is to incorporate checking into the programs of the computer. The first approach, programmed "software checks," is the responsibility of the programmer and does not affect the logic design of the arithmetic unit. The second approach, "hardware checking," has additional hardware introduced into the computer, and the result of each arithmetical algorithm is automatically (without programmed commands) tested for acceptability. Hardware checking relieves the programmer of the chore of programming checks; consequently, it may be expected to achieve the acceptance found by hardware implementations of floating-point arithmetic and by the more recent hardware algorithms for significant digit arithmetic, both of which are replacing programmed implementations. The general acceptance of hardware checking depends on the development of relatively economical and fast algorithms for the checking operations. The self-evident method of duplicating the arithmetic unit offers a reference point for a search for more economical approaches.

C. *Properties of Arithmetical Errors*

To limit the scope of this investigation, attention will be restricted to arithmetical errors, leaving control errors as a separate problem of checking control operations throughout the computer. A study of arithmetical errors requires, first of all, a complete enumeration and classification of these errors.

Given a computer which employs conventional constant base b digital numbers n digits long, an arithmetical error occurs if the **correct** (i.e., specified) result corresponding to (OC, X, Y) is the digital number S composed of n digits s_i: $s_{n-1} \ldots s_i \ldots s_0$, and the **actual result** is the digital number S^* such that

$$s_i \neq s_i^* \qquad (0 \leq i \leq n-1)$$

holds for one or more positions i of S and S^*.

Corresponding to every possible arithmetical error an **error number** E, is defined as composed of n digits: $e_{n-1} \ldots e_i \ldots e_0$, such that

$$e_i = s_i^* - s_i \qquad (0 \leq i \leq n-1) \tag{1}$$

The error number E is a redundant form, since its digits may assume any one of $2b-1$ values

$$-(b-1), \ldots, -1, 0, 1, \ldots, (b-1)$$

when both s_i and s_i^* range from 0 to $b-1$.

An arithmetical error corresponds to an undesired addition of the error number E (defined in terms of its digits e_i) to the correct result S such that the actual result

$$S^* = S + E \tag{2}$$

is generated, or delivered from storage, by the arithmetic unit.

The most significant properties of arithmetical errors are their detectability and their probability of occurrence. These properties will be investigated by a study of error numbers E which assume the integer values

$$E = \sum_{0}^{n-1} e_i \, b^i \tag{3}$$

Since there are $2b-1$ possible values for each e_i $(0 \leq i \leq n-1)$, there exists a total of

$$N = (2b-1)^n - 1 \tag{4}$$

different forms of error numbers E (excluding the case $E = 0$). The N different forms are divided into $N/2$ symmetrical pairs of forms. In each pair one form has the value $+|E|$, and the other form has the value $-|E|$. One member of the pair is obtained from the other by changing the signs of all nonzero digit values ($e_i \neq 0$).

The range of the values of E does not exceed the range which is representable by the digits s_i of S, so that

$$(b^n - 1) \geq |E| \geq 1 \tag{5}$$

is the range for the magnitudes of E. There is at least one pair of forms corresponding to every possible magnitude $|E|$. Consequently, all $(2b-1)^n - 1$ possible forms of error numbers E are classified into $b^n - 1$ **magnitude classes** according to the value of $|E|$, and every arithmet-

ical error is said to have the magnitude $|E|$, which is the magnitude of the associated error number E, computed as

$$\left| E \right| = \left| \sum_{0}^{n-1} e_i \, b^i \right| \tag{6}$$

In all known methods of numerical checking for digital arithmetic, the detectability of an arithmetical error depends on its magnitude. Because of the redundancy of the error numbers E, most magnitude classes will contain more than one pair of different forms. These pairs of forms will differ in the values, the locations and the total number of nonzero digits, e_i.

The second important property of an error number E is its **multiplicity** μ, which is defined as the count of nonzero digit values ($e_i \neq 0$) in a given form of E. In every magnitude class there will exist one or more pairs of forms which have the least value of μ among all members of that magnitude class. This value is the minimal multiplicity or **weight** of the given error magnitude $|E|$. The weight indicates the least number of independent changes (not dependent on carry propagation) in the digits of the operands or the results which are necessary to cause an error of magnitude $|E|$ during an addition. Because of carry propagation, the error number E may assume a form which is not of minimal multiplicity.

In summary, every symmetrical pair of error numbers has an associated error magnitude $|E|$ and an error multiplicity μ. Every magnitude class $|E|$ (containing one or more of these pairs) has a minimal multiplicity, called its weight, $w(|E|)$. The value of an error magnitude indicates its detectability, while the weight of an error magnitude may indicate the relative probability of its occurrence. Further consideration of this probability is necessary because repetitive use of a single defective circuit generally will not yield an error of weight 1.

D. Errors in Binary Arithmetic

The preceding definitions and properties of arithmetical errors apply equally well to any conventional number system with a constant positive base $b \geq 2$ and may be readily extended to other positional number systems. For a more detailed investigation attention will be focused on base 2, the case of most immediate interest.

For binary numbers of n-digits-length, the value of the error number is

$$E = \sum_{0}^{n-1} e_i \, 2^i, \qquad (e_i = 0, +1, -1) \tag{7}$$

The error number contains $e_i = +1$ for every position i in which the correct digit $s_i = 0$ was replaced by the incorrect digit $s_i^* = 1$, and $e_i = -1$ for a change from $s_i = 1$ to $s_i^* = 0$. There exists a total of $3^n - 1$ distinct forms of E, which are divided into $2^n - 1$ magnitude classes, since $2^n - 1 \geq |E| \geq 1$ is the range of n-digit binary numbers.

For any given error magnitude $|E| = K$ it is necessary to establish the weight (minimal multiplicity) $w(K)$. The definition of the multiplicity μ for a binary error number E with digits e_i is

$$\mu(E) = \sum_{i=0}^{n-1} |e_i| \tag{8}$$

Among all t pairs of forms $\{\pm E_{K1} \ldots, \pm E_{Kj} \ldots, \pm E_{Kt}\}$ which belong to the magnitude class K there is a **minimal form** $\pm E_{KM}$ for which

$$\mu(\pm E_{KM}) = w(K) \leq \mu(\pm E_{Kj}), \text{ for } t \geq j \geq 1 \tag{9}$$

The existence and generation of such minimal forms has been investigated for the recoding of multipliers in accelerated multiplication (Ref. 1). Using these results, the weights $w(K)$ may be computed for all $2^n - 1$ error magnitudes. According to the theory of multiplier recoding, a binary number E possesses the "unqualified property M" if and only if

$$e_i \, e_{i-1} = 0, \text{ for } \qquad 1 \leq i \leq n-1 \tag{10}$$

which means that no two adjacent digits of E are both nonzero. After imposing the further restriction that

$$e_{n-1} \, e_{n-2} \neq +1 \tag{11}$$

(i.e., the two leftmost digits of E are not both $+1$ or both -1), Reitwiesner (Ref. 1) shows that

1. For any error magnitude K there exists among its forms a pair of unique forms $\pm E_{KM}$ which possess the unqualified property M;
2. No other form in the magnitude class K has a lower multiplicity than the multiplicity $\mu(E_{KM})$ of $\pm E_{KM}$.

The restriction $e_{n-1}\, e_{n-2} \neq +1$ can be imposed in the study of multiplier recoding because of the limited range of multipliers. This restriction does not apply to binary error numbers E, for which $e_{n-1}\, e_{n-2} = +1$ is also possible; therefore, minimality must be defined for this case.

The error number E possesses the "restricted property M" if and only if

1. $e_i\, e_{i-1} = +1$ holds pairwise for $n-1 \geq i \geq k$; that is, the digits $e_{n-1}, e_{n-2}, \ldots e_k$ all have the same value, $+1$ or -1;
2. $e_i\, e_{i-1} = 0$ holds for $k \geq i \geq 0$; this means that the digital number composed of the rightmost digits $e_k, e_{k-1}, \ldots, e_0$ possesses the unqualified property M. Since $e_k = +1$ or -1, $e_{k-1} = 0$ will always hold.

The existence and minimality proofs of the unqualified property M extend readily to the restricted property M.

The minimal forms $\pm E_{KM}$, corresponding to any error magnitude K, are defined to be the unique pair of forms which possess the (unqualified or restricted) property M among all forms in the magnitude class K. The multiplicity $\mu(\pm E_{KM})$ of this pair is defined as the weight $w(K)$ of the error magnitude K.

Given any one form $+ E_{Kj}$ from the set of forms corresponding to error magnitude K, the minimal form $+ E_{KM}$ is generated according to the recoding rules given by Ref. 1. All other forms of magnitude K may be generated from the minimal form $+ E_{KM}$ by reversing the recoding procedure. The definitions and rules given in this section provide practical methods for the determination of the weights $w(K)$ and for the enumeration of all forms belonging to any magnitude class K.

E. The Effects of a Circuit Failure in Binary Arithmetic

Arithmetical errors occur when a defective component is employed in generating the result of a specified algorithm. The external effect of a defective component is observed as the failure of a logical circuit to perform its prescribed function. The two principal modes of failure of logical circuits are termed "stuck on 0" and "stuck on 1". These terms apply to four different locations:

1. an input to a binary storage circuit;
2. an input to a combinational logic circuit (gate);
3. an output of a binary storage circuit;
4. an output of a combinational logic circuit (gate).

If the defective circuit is employed only once in a particular algorithm, the effect of the failure is the replacement of a single binary variable (0 or 1) by its complement (1 or 0, respectively). Physically, the replacement may occur in the input operands, at an input or output of combinational logic, or in the result itself. The effect of the failure will be further localized if the design of the arithmetic unit is such that one use of a failed circuit can affect only one binary digit. This requires, for instance, that in a binary adder the circuits which generate the sum digit should be independent of the circuits which generate and transmit carries.

When the above given conditions are satisfied, an error number E_j will be added to the correct result by a single use of a failed circuit which affects the j-th digit of the n-digit result. The actual result will be

$$S^* \equiv (S + E_j) \text{ modulo A} \tag{12}$$

in an arithmetic unit which employs a modulo-A adder or subtractor. The four possible nonzero values of E_j in this case will be (with $n-1 \geq j \geq 0$):

$$E_j = 2^j$$

$$E_j = -A + 2^j$$

$$E_j = -2^j$$

$$E_j = A - 2^j$$

The first two values occur when a unit is added in position j during the execution of the algorithm. (The second case includes a subtraction of A due to the error.) The last two values occur when a unit is subtracted in position j. (The fourth case includes a failure to subtract A due to the error.) A fifth case

$$E_j = 0$$

will occur when the error condition is the same as the desired value. In this case the failure will have no observable effect and will not change the correct result. It is evident that the nonzero errors E_j will belong to two magnitude classes, which are

$$|E_j| = 2^j$$

$$|E_j| = A - 2^j$$

with j in the range $n-1 \geq j \geq 0$ for n-digit numbers.

In a completely parallel operation, such as parallel transmission, digitwise complementation of a register, or a single shift, the actual error number will contain only a single digit of value $+1$ or -1. This condition will also hold in parallel addition of two digital numbers if the error is caused by the circuits which form or store the sum digits. If the error of magnitude $\pm 2^i$ is introduced in the input operands, or if it is caused by the carry-generation or carry-propagation logic, the corresponding error number may contain a string of ∓ 1 digits, beginning at the position i and terminating at some position $i + k$ with a ± 1 digit. For instance, a five-bit error number with the value $E = +1$ may assume the following five forms: 00001, 0001$\bar{1}$, 001$\bar{1}\bar{1}$, 01$\bar{1}\bar{1}\bar{1}$ and 1$\bar{1}\bar{1}\bar{1}\bar{1}$, after an addition, while only the first form will occur after a completely parallel operation. When the addition is performed modulo $2^5 = 32$, the form $\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}$ (value -31) may also occur, while for addition modulo $2^5 - 1 = 31$, the form $\bar{1}\bar{1}\bar{1}\bar{1}0$ (value -30) caused by the "end around" carry can be found. For the last two cases the relationships are: $+1 \equiv -31$ modulo 32, and $+1 \equiv -30$ modulo 31.

F. The Effects of a Repetitive Use of One Defective Circuit and of Multiple Failures

In many cases of practical interest the circuits of the arithmetic unit are used more than once during a given algorithm. Serial shifting of several positions and serial addition serve as examples of such repetitive use, as well as multiplication and division in any arithmetic unit. Evidently, a single defective component, when used repetitively during an algorithm, will generate an error number whose magnitude in most cases differs from 2^i. If errors of magnitude 2^i (weight 1) are called single errors, then a single defective circuit may generate other than single errors.

Since the practical objective of error detection and/or correction is the protection against, at least, single failures of circuitry, it is necessary to investigate the error numbers generated by repetitive use of a defective circuit during one algorithm. A thorough knowledge of their properties, in turn, may indicate which variations of the arithmetical algorithms will yield the highest probability of single-failure detection.

A more detailed definition of circuit failure is necessary in the case of repetitive use of defective circuits. For single use it is sufficient to state that an inversion of a binary signal occurs (0 instead of 1, or 1 instead of 0 is generated). For repetitive use, however, both the duration and the mode of failure must be considered. A **failure** has occurred if the mode of failure persists for the entire duration of the algorithm (i.e., for all uses of the circuit); if the mode of failure does not last for all uses of the circuit, a **malfunction** has occurred.

The two modes of failure considered here are "stuck on 0" and "stuck on 1". The damage which is inflicted on a result by repeated use of the defective logic circuit is called an erasure. An **erasure** is described as follows: given a set of positions $\{j\}$ in a binary number, which are occupied by binary digits of value 0 or 1, a **down-erasure** occurs when all these digits are replaced by 0's, and an **up-erasure** occurs when they are replaced by 1's. An erasure is specified completely by the list of positions and by the direction (up or down). In the case of a failure, the positions of an erasure depend on the details of the algorithm. Unless special precautions are taken, these positions may be a continuous string over a part of the result or over the entire result. In the case of a malfunction, the positions of an erasure also depend on the duration and timing of the malfunction. Only a part of the previous set of positions (affected by a failure) will be affected by a malfunction.

An error number of value E_j corresponds to every position of an erasure. An erasure number has the value

$$E' = \sum E_j \text{ modulo A}$$

where all error numbers E_j for the given set $\{j\}$ of erasure positions are included in the sum. For the mode "stuck on 1", the values of E_j may be 2^j, $-A+2^j$, and zero. Then

$$E' = \sum 2^j, \text{ or } E' = -A + \sum 2^j$$

For the mode "stuck on 0", the values of E_j may be -2^j, $A - 2^j$, and zero. In this case

$$E' = \sum -2^j, \text{ or } E' = A - \sum 2^j$$

These values form the set of potential values of an erasure number. It is necessary to observe that in the definition of an erasure every erasure position j can be affected only once by the failed circuit. If this restriction is not satisfied, one failure may add two or more erasure numbers to the result.

A third mode of failure, which has not been considered before, is "stuck on X". In this case the value X is indeterminate in terms of 0 and 1 and may be interpreted randomly as either 0 or 1 by the following logic circuits. In this case the value of the erasure number

$$E' = \sum E_j \text{ modulo A}$$

may have any one of the five values $\pm 2^j$, $\pm (A-2^j)$ and zero for every E_j in the sum, and the set of potential values is further expanded.

It should be observed here that an erasure due to the failure mode "stuck on X" has the same effect as several independent failures during an algorithm in which every failed circuit is used only once. The values of the error number (or erasure number) due to a failure of two or more logic circuits are obtained by summing the error (or erasure) numbers due to every separate failure. The probability of any particular error magnitude can then be estimated by using combinational mathematics.

II. HARDWARE METHODS FOR THE DETECTION OF ARITHMETICAL ERRORS

A. Introduction

Failure of a single component in an arithmetic unit may produce an incorrect result which invalidates the entire programming effort and wastes computing time. Arithmetical errors may be detected either by means of programmed checks or by means of hardware checks (additional hardware associated with the arithmetic unit). Hardware checking relieves the programmer of the nonproductive effort of programming checking operations. Furthermore, hardware checking is applied to the result of every arithmetical algorithm which corresponds to a single instruction. A component failure or temporary malfunction is detected immediately after its occurrence, and corrective action may be initiated with a minimal loss of time, employing a program-independent interrupt sequence.

Two types of corrective action may be initiated after an arithmetical error has been detected. One type is automatic error correction, which is often implemented in the transmission of information by means of error-correcting codes and special hardware for error correction. The other type is a replacement sequence or a repair action which is initiated upon the detection of an arithmetical error. Automatic error correction is necessary for table-lookup type arithmetic units in which all results are stored permanently, and data transmission from storage is the only operation. Error-correcting codes, which have been devised for data transmission, are applicable in this case.

When computation in the arithmetic unit is by means of algorithms, automatic error correction implies a second arithmetic unit which computes corrections for the incorrect results of the first. Furthermore, the error-correcting codes used in transmission of information do not retain their properties when subjected to arithmetical algorithms; and, consequently, new codes must be devised. The replacement or repair of a defective arithmetic unit is, therefore, the more practical solution for algorithm-type arithmetic units. The entire attention in this case may be focused on codes and methods of detecting arithmetical errors.

B. Methods of Numerical Checking

In algorithm-type arithmetic units the elementary algorithms are those used for transmission, digitwise complementation, addition (and/or subtraction), and shifting of digital numbers. Compound algorithms are executed as sequences of these elementary algorithms. Consequently, checking of the elementary algorithms is a prerequisite for any error detection scheme. The checking of data transmissions also provides a method for checking the validity of input operands, which are transmitted from other parts of the computer.

We have observed that an arithmetical error may be described as the addition of the error number E to the correct result S, such that the actual result $S^* = S + E$ is generated. In conventional number systems every out-

put result is a valid digital number, and, consequently, S and S^* cannot be distinguished for any value of E. Evidently, additional information must be introduced into the number system which will permit us to distinguish S and S^* for at least some (most likely) values of E.

Two methods have been proposed for checking addition and other elementary algorithms:

1. The attachment of **check symbols** to all digital numbers in the computer.
2. The **arithmetical encoding** of all digital numbers in the computer.

The objective of both methods is to provide the means to distinguish S and $S^* = S + E$ for a chosen set of error magnitudes $\{|E|\}$. Both methods depend on the additive character of the arithmetical errors and differ principally in the implementation of the checking algorithm.

Since the effects of all elementary algorithms may be described in terms of an addition, checking the SUM algorithm is the principal problem. For example, the other base 2 algorithms are included by considering the transmission of X as the addition $X + 0$, the left shift of X as the addition $X + X$, the complementation with respect to A as the addition $A+(-X)$, and the right shift of X as the addition $X+(-X/2)$. Multiplication and division are composed of sequences of the elementary algorithms.

C. *Independent Adder and Checker System*

The first important step in devising checking procedures for an arithmetic unit is the choice of a checking procedure for the SUM algorithm, that is, for the output of an adder, several of which have been suggested or investigated in current technical literature.

One method of checking an adder employs an independent checker. The adder and the checker are two completely independent circuits. Each digital number X of a given system has a check symbol $c(X)$, which is stored as a pair $[X, c(X)]$ with the number. When two numbers (X,Y) are sent to the adder to form the sum $X + Y$ their check symbols are sent to the checker, which forms the output $c(X)*c(Y)$. This checker output must be the check symbol for the sum; that is, the following requirement is to be satisfied

$$c(X+Y) = c(X)*c(Y) \qquad (13)$$

An acceptance check must be performed to test whether Eq. (13) is satisfied. This check consists of computing $c(X+Y)$ from the adder output $X+Y$ and comparing it with the checker output $c(X)*c(Y)$. If Eq. (13) is not satisfied, a computational error has occurred either in the adder or in the checker circuits. If Eq. (13) is satisfied, the addition is accepted as valid, although an undetectable error may have occurred. A block diagram of the checking system is shown in Fig. 1.

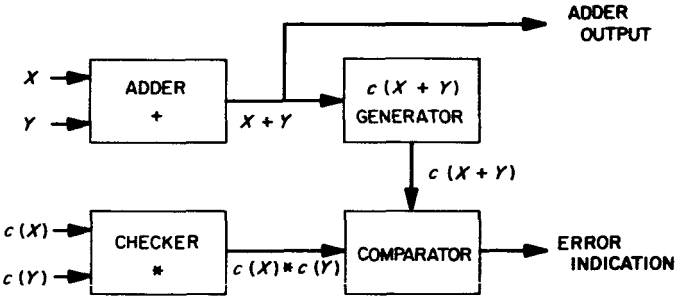

Fig. 1. Adder with independent checker

The independent checker system has been studied by W. W. Peterson (Ref. 2 and 3). In these references he discusses independent checkers and proves a basic theorem on such systems.

> Peterson's Theorem: If there are fewer check symbols than integers in the allowed range of the digital numbers, and if the check symbols satisfy Eq. (13), then $c(X)$ must be the residue of X modulo some integer β in coded form, where β is the number of distinct check symbols; and the operation * is addition modulo β.

Evidently, if there are as many check symbols as integers, each integer has a separate check symbol, and the checker is a duplicate adder.

The independent checker system has also been studied by H. Garner (Ref. 4), who discusses application of the check moduli $(b+1)$ and $(b-1)$ for base b number systems. An early design [1948–1950] of a digital computer employing this type of checking is described in Ref. 5 and 6. The RAYDAC computer, which had an independent checking arithmetic unit (discussed in Ref. 5 and 6) was recently decommissioned after almost a decade of satisfactory operation.

Generally, an independent checker system is limited to the detection of errors, since there is no indication

whether the failure occurred in the adder or in the checker. The undetectable errors correspond to all error words E such that

$$|E| = k\beta \equiv 0 \bmod \beta \tag{14}$$

that is, the magnitude of E is an integral multiple k of the check modulus β. (k and β are positive integers.) A less probable undetectable error occurs when both the adder and the checker fail simultaneously and indicate acceptance, or when the comparator fails.

Since the operation of the checker is addition modulo β, a residue number system of range $0 \leq c(X) < \beta$ may be conveniently chosen as the set of check symbols. A limited range residue arithmetic unit then serves as the "checking arithmetic unit".

D. Arithmetical Encoding of Digital Numbers

Another method for checking the SUM algorithm is the premultiplication of all digital numbers X in the allowed range by an integer constant α, so that all numbers Z entering the adder are encoded in product form ($Z = \alpha X$). Because of the distributive law of algebra,

$$\alpha X + \alpha Y = \alpha (X+Y) \tag{15}$$

that is, the output of the adder should be a digital number in properly coded form. To test the output for validity, divide it by α and inspect the remainder. A nonzero remainder (the least positive residue of the adder output, modulo α) indicates that an arithmetical error has occurred and that the adder output is actually

$$S^* = \alpha (X+Y) + E \tag{16}$$

where the error number E satisfies

$$|E| \neq k\,\alpha \text{ for } k,\ \alpha \text{ positive integers.} \tag{17}$$

All errors corresponding to error numbers E, such that $|E| = k\alpha$, will remain undetected by this checking procedure. It is evident that both the quantity and the multiplicity of undetectable error numbers depend on the choice of α.

Considering the possible values of α, it is observed that if α has the base b of X as a factor ($\alpha = kb$), then the right end digit of αX is always zero and, therefore, useless. All allowed values of this right end digit will occur only if α and b are relatively prime, and this requirement should be satisfied for efficient utilization of storage.

To detect any weight-1 error (only one digit $e_i \neq 0$ in the error number E) the requirement

$$e_i\, b^i \neq k\alpha \tag{18}$$

should be satisfied by α for any positive integer k and for $|e_i| < b$. Equation (18) will be satisfied by any $\alpha > b$, which is also relatively prime to b. The smallest value of α which satisfies these requirements is

$$\alpha = b+1 \tag{19}$$

This value requires not more than two extra digits to encode X into αX.

The detection of errors of higher multiplicity and, also, the correction of errors has been investigated for base $b = 2$ in the following references. The earliest description of product encoding for error detection is attributed to J. M. Diamond (Ref. 7). Later D. T. Brown (Ref. 8) described a class of double-error detecting and single-error correcting binary codes. This work was further systematized and some codes for triple-error correction were described by W. W. Peterson (Ref. 2).

Considering the product encoding method more generally, an arithmetical transformation is applied to each digital number X to get the coded form $\phi(X)$. The adder forms the sum $Z = \phi(X) + \phi(Y)$ which should satisfy the requirement

$$\phi(X) + \phi(Y) = \phi(X+Y) \tag{20}$$

The acceptance test is made by testing whether the adder output Z is a member of the coded set of numbers; this may be considered as the application of a reverse transformation ϕ^{-1} to Z. In the previous method ϕ was multiplication by α, and ϕ^{-1} was division by α. A block diagram of the general coding method is shown in Fig. 2. An error indication occurs if adder output Z is not a properly coded number.

Beside product encoding (premultiplication) discussed above, a sum encoding has been suggested, in which a

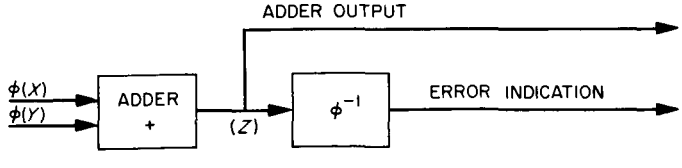

Fig. 2. Adder for encoded operands

k-digits-long check number $\#(X)$ with β distinct values is computed and added to the n-digits-long operand X, such that either

$$\#(X)\, b^n + X \equiv 0 \quad \text{modulo } \beta \tag{21}$$

is satisfied by the composite number with $\#(X)$ attached at the left end of X, or

$$X\, b^k + \#(X) \equiv 0 \quad \text{modulo } \beta \tag{22}$$

is satisfied by the composite number with $\#(X)$ attached at the right end of X.

Both schemes have been only sketchily suggested, Eq. (21) in Ref. 9 and Eq. (22) in Ref. 2, page 244. The problems of carry-transmission between X and $\#(X)$ during addition, and especially those in multiplication and division have not been solved for such sum encoding.

E. Merit Criteria for the Check Moduli

In both methods of checking (separate check symbols and product encoding) the undetectable errors correspond to all error numbers E whose magnitudes satisfy the equation

$$|E| = k\alpha \tag{23}$$

where the check modulus α and the coefficient k are both positive integers. The remaining problem is to select a value of α which is best suited for checking a given arithmetic unit.

The currently existing criterion for the choice of α is the elimination of all errors with low values of the weight (minimal multiplicity) of corresponding error numbers (Ref. 2 and 8). Values of α have been determined which yield codes of specified distances $d=2$, $d=3$ and $d=4$ for binary numbers X in the range $0 \leq X < M_2\ (\alpha, d)$. A code of distance d detects all errors of weight $d-1$ and less when the range of the (uncoded) numbers given above is not exceeded by the results of the addition. The value of $M_2\ (\alpha, d)$ is a function of α and the specified distance d.

It is proposed here that a second criterion, namely the cost corresponding to the given choice of α, should be considered in the choice of α. It is argued that by permitting a small percentage of previously detectable (say, double) errors to remain undetected, the cost and the probability of failure of the checking algorithm may be decreased by a significant amount. Instead of "specified detection at any cost" the more practical approach "effective detection at low cost" is proposed.

The cost incurred by introducing error-detecting algorithms into an arithmetic unit is distributed as follows:

1. the cost of the special checking hardware;
2. the increase in the length of digital numbers, requiring additional storage capacity;
3. the increase in the duration of the algorithms in the arithmetic unit;
4. the increase in the complexity of the original arithmetic unit and arithmetic control.

It may be expected that the cost will vary for various choices of α. From the viewpoint of cost, choices of α which give the least increase in cost [expressed as some cost function $C(\alpha)$] are most desirable.

The original distance criterion may be more generally stated in terms of the probability $P(\alpha)$ that an erroneous result will remain undetected for a given value of α and for a given set of arithmetical algorithms with their hardware implementation. In the discussion of arithmetical errors it may be recalled that erasure errors may result from a single defective circuit in some algorithms, for instance in multiplication and division. To determine $P(\alpha)$ for a given set of algorithms, it is necessary to express $P(\alpha)$ as the sum of probabilities that an undetectable error of magnitude $|E| = k\alpha$ will occur, given a certain probability of failure for the circuits of the arithmetic unit.

In summary, a systematic approach to the choice of the optimum α for a specified arithmetic unit requires the consideration of both $P(\alpha)$ and $C(\alpha)$ for values of α which are acceptable from viewpoints of both cost and error detection probability. The simultaneous minimization of properly weighted values of both $P(\alpha)$ and $C(\alpha)$ over the acceptable range of values of α may be expected to yield a practically acceptable design for an arithmetic unit with built-in error detection. The remaining significant problem is the assignment of the cost function $C(\alpha)$ and of the undetected error probability function $P(\alpha)$ to a given arithmetic unit. A second problem of great interest is the choice of algorithms which yield the lowest values of $P(\alpha)$ and $C(\alpha)$ among various possible implementations of an arithmetic unit.

III. MINIMAL COST CHECKING ALGORITHM FOR THE DETECTION OF ARITHMETICAL ERRORS

A. Introduction

Effectiveness and cost are the two principal criteria which determine the usefulness of a scheme for detecting arithmetical errors. A reference point for effectiveness is the set of specified distance d (d=2, 3, 4) codes which were discussed in the preceding chapter. For these codes the specified distance is attained, regardless of cost, by the choice of the check modulus α. The codes have been discussed with respect to addition (the SUM algorithm); the implementation of checking for an entire set of arithmetical algorithms remains to be developed.

The cost is proposed in this investigation as the second criterion for determining the usefulness of a given check modulus α. To serve as a reference point for cost investigations a set of minimal cost algorithms for an arithmetic unit with error detection of acceptable effectiveness is developed. For this purpose it is necessary to postulate a typical arithmetic unit and to consider the costs of incorporating error detection.

The choice of a typical arithmetic unit is governed by the practical aspect of this investigation: the development of an arithmetic unit for a redundant replacement system which would serve as a self-testing and repairing guidance computer for space vehicles. This practical application of the checking algorithms directs the choices among many alternatives in the specification of the typical arithmetic unit. A summary of these alternatives is presented below.

B. Alternate Choices for Arithmetical Checking

A series of choices which affect error checking must be made in the selection of the characteristics of a particular arithmetic unit. The initial choice is between an algorithmic and a storage (table-lookup) arithmetic unit. The latter may employ transmission type error detecting/correcting codes, while the former requires arithmetical checking. The objective of arithmetical checking may be either automatic error correction, or error detection followed by replacement or repair of the defective arithmetic unit. The estimated lowest hardware cost indicates the algorithmic arithmetic unit with error detection as the first choice for the intended application.

Both arithmetical coding and separate check-symbols may be used for error detection. Arithmetical coding offers the advantage of a uniform code-protected number system throughout the computer with the possibility of error correction in the operands brought up from the memory or from input buffers. Furthermore, less separate checking hardware is required when separate operations on the check symbols are eliminated.

Product coding is the only developed scheme of arithmetical coding. It is suitable for checking addition because of the distributive law: $\alpha X + \alpha Y = \alpha(X+Y)$. The remaining choices are the base of the coded number system and the value of the check modulus α. Base 2 is preferable because of its simplicity and its direct relationship of circuit failures to changes in digit values. The choice of α remains to be discussed below.

In summary, the preferred choice is an algorithmic binary arithmetic unit with error detection implemented by product coding all digital numbers throughout the computer. To isolate the effects of single circuit failures, a parallel adder with separate sum and carry circuits is preferred. For a review, the alternatives for arithmetical checking which were considered are summarized in Fig. 3.

C. The Cost of Checking Algorithms

The error-detecting (decoding) algorithm for arithmetical results computed from product coded digital numbers Z ($Z = \alpha X$) computes the remainder R resulting from a division of the result S by α. The value $R=0$ indicates that the digital number S is properly coded and acceptable; any other value of R ($\alpha - 1 \geq R \geq 1$) indicates that result S contains an arithmetical error. Transmission errors are included as a special case of arithmetical errors.

The algorithm may be implemented either in the arithmetic unit itself, preceding and/or following an arithmetical operation, or in an independent checker whose only function is the decoding algorithm. The use of an independent checker is preferable because the separate checker is not affected by failures of the arithmetic unit. Concurrent operation with the arithmetic unit is a

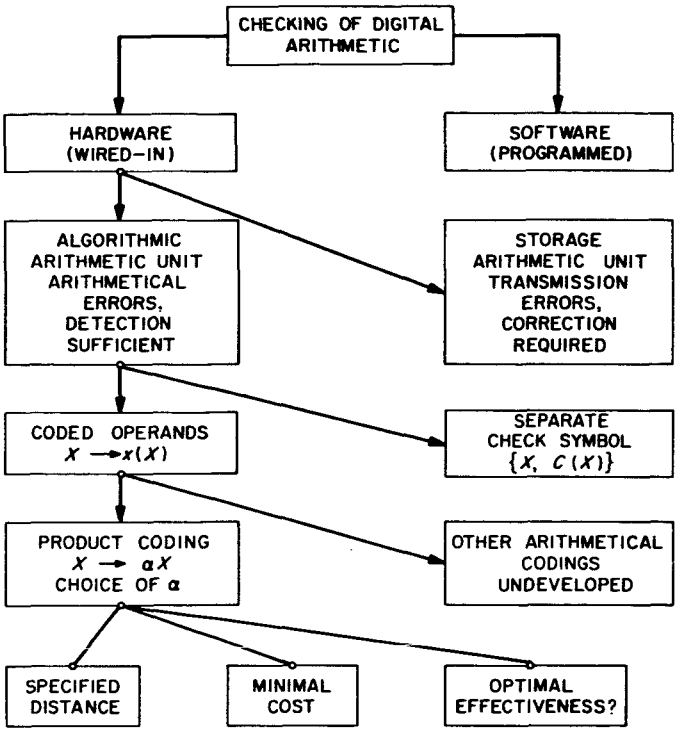

Fig. 3. Checking methods for digital arithmetic

further advantage of the separate checker. Consequently, the cost and operation time of the checker for a given α is one measure of the cost associated with this α.

The implementation of the decoding algorithm requires hardware for a division of the result S by the constant α, in which the quotient is not retained, and a subsequent test of the remainder R for the condition $R=0$. Given an n-digit binary number S the required hardware is an n-digit shift register and a subtractor for subtracting α at the high-significance end of the shift register. Given an a digit binary constant α $(\alpha_{a-1}, \ldots, \alpha_0)$ the subtractor requires one full stage for every 1 except α_0 and one-half stage for every 0 and for α_0. Alternatively, $(2^a - \alpha)$ may be added modulo 2^a to the a leftmost digits of the S register, if this operation is more economical. A circuit which indicates when the subtraction is to be performed (i.e., when the quotient digit is 1) is also required. Any of the existing division algorithms which yield a correct remainder may be employed; the quotient digits are not retained.

The cost of the decoder may be estimated from the count of 0 and 1 digits in α and $2^a - \alpha$, allowing one full adder (or subtractor) stage for every 1 except α_0 and a half-stage for every 0 and for $\alpha_0=1$ (since α is odd, $\alpha_0=1$ always holds). The time requirement is $n-a$ shift-times, plus the time for choosing quotient digits and the time for testing the remainder for $R=0$. In error-correcting codes, the nonzero value of R also conveys information on the value of the error number.

D. The Minimal Cost Checker

An exception in the implementation of the division S/α occurs for the special choice of

$$\alpha = 2^a - 1; \text{ for } a \geq 2 \tag{24}$$

In this case the identity

$$Kr^i \equiv K \text{ modulo } (r - 1) \tag{25}$$

where K, r, and i are positive integers, is employed. Choosing $r = 2^a$ and relabeling $K = t_i$ gives

$$t_i (2^a)^i \equiv t_i \text{ modulo } (2^a - 1) \tag{26}$$

Now any base-2 integer S with n digits s_i may be considered to be a base-2^a integer T with k digits t_i $(0 \leq t_i \leq 2^a - 1)$, where $n = ka$. Consequently, the value of S is

$$S = \sum_0^{n-1} s_i 2^i = \sum_0^{k-1} t_i (2^a)^i \equiv \sum_0^{k-1} t_i \text{ modulo } (2^a - 1) \tag{27}$$

The remainder of S upon division by $2^a - 1$ can be computed as the least positive residue of S modulo $2^a - 1$, denoted $|S|_{2^a-1}$, by adding the digits t_i modulo $2^a - 1$. Since every digit t_i consists of a binary digits, this is the addition of a-digits-long sections of S with an "end-around" carry.

The minimum cost implementation of this algorithm requires an n-digit shift register $(n = ka)$ with n flip-flops $(n - 1$ to $0)$ and one full binary adder stage, as shown in Fig. 4. No decision on quotient digit value is required.

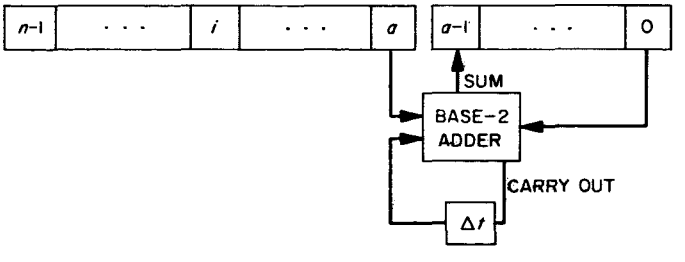

Fig. 4. Minimum cost checker for $\alpha = 2^a - 1$

S is stored in the shift register, and initially the flipflops $a-1$ to 0 of the shift register hold the digit t_0. The entire register shifts right, with the digits in flipflops a and 0 entering the adder and the sum digit entering the flipflop $a-1$. After $n-a$ right shifts the flipflops $a-1$ to 0 and the carry stored in delay Δt represent the remainder R

$$R = \left| S \right|_{2^a-1} = \left| \sum_{0}^{k-1} t_i \right|_{2^a-1} \tag{28}$$

and may be examined for the condition $R = 0$.

Faster implementations of the checking algorithm are possible. If the single base-2 adder stage is replaced by a parallel adder of a-bits length, the register shifts a bits at once, and the addition to completed after $k - 1 = (n/a) - 1$ shifts. Faster addition may be performed by employing a cascaded arrangement of carry-save adders.

In summary, the check modulus $\alpha = 2^a - 1$ is observed to have special properties which yield the lowest cost for the implementation of the checker. Only one full adder is required and special sensing circuitry (needed to determine the quotient digit) is eliminated for any $a \geq 2$. An increase in speed may be gained by additional hardware, since the division algorithm is replaced by a summation of several a digits long sections of a binary number. A further advantage of $\alpha = 2^a - 1$ is that the value $2^a - 1$ is readily tractable in binary arithmetic and may be expected to yield economical algorithms for coded operands in the arithmetic unit.

E. The Effectiveness of a Check Modulus

Given the integers X to be product-coded as $Z = \alpha X$ in the range $B > X > 0$, the undetectable error numbers will have the magnitudes

$$|E| = C\alpha; \text{ with } B > C > 0 \tag{29}$$

When the uncoded number system employs complements with respect to B to represent negative integers, then

$$A - \alpha X = \alpha (B - X); \text{ i.e., } A = \alpha B \tag{30}$$

is required to hold for the product-coded number system in which negative integers are represented as complements with respect to A. In this case, if the error magnitude $|E| = C\alpha$ is undetectable, then

$$A - |E| = \alpha (B - C) \tag{31}$$

is also undetectable.

The next question of immediate interest is the effectiveness of the class of check moduli $\alpha = 2^a - 1$ with $a \geq 2$. The expression $2^a - 1$ is relatively prime to the base $b=2$ (and any $b=2^i$), but no values of $\alpha = 2^a - 1$ are listed among the product codes of distance $d \geq 3$. Thus, $\alpha = 2^a - 1$ will yield only a distance $d=2$ (single-error detecting) code with distance as an absolute criterion. The undetectable errors for an adder operating modulo $A=B\,(2^a-1)$ have only the possible magnitudes

$$|E| = C\,(2^a - 1), \text{ for } B > C > 0 \tag{32}$$

These magnitudes correspond to all valid coded digital numbers $Z = (2^a - 1)X$ in the range $A > Z > 0$.

The effectiveness of $\alpha = 2^a - 1$ may be estimated by considering how many error numbers of weight (minimal multiplicity) 2, 3, etc. will remain undetected among all possible error numbers of the given weight. All error numbers of weight 1 will evidently be detected, since $C\,(2^a - 1) \neq 2^i$ for a > 1. A further study of erasure errors is necessary for algorithms in which a defective circuit may be employed more than once.

F. Detection of Error Magnitudes of Weight 2

Next it is established how many and which error magnitudes of weight 2 remain undetected when $\alpha = 2^a - 1$ is used as a check modulus. The coded numbers $Z = \alpha X$ consist of n binary digits $(z_{n-1} \cdots z_i \cdots z_0)$ and have the range

$$2^n - 1 \geq Z \geq 0 \tag{33}$$

According to the theory of minimal forms (discussed previously), the error magnitudes of weight 2 will be of the form

$$|E| = 2^j \pm 2^i \quad \text{for} \quad n - 1 \geq j > i \geq 0 \tag{34}$$

excluding $j = i + 1$ (i.e., adjacent ones), but not excluding the restricted minimal form $|E| = 2^{n-1} + 2^{n-2}$.

To count magnitudes $|E| = 2^j + 2^i$ (with $j \neq i + 1$), the form is restated as $|E| = 2^i\,(2^m + 1)$ with $n - 1 \geq m \geq 2$ and $(n-1) - m \geq i \geq 0$. Counting the multiples $2^i\,(2^m+1)$ for every value of m, there are

$n-2$ multiples for $m = 2$, $(n-3 \geq i \geq 0)$

$n-3$ multiples for $m = 3$, $(n-4 \geq i \geq 0)$

. .

. .

. .

1 multiple for $m = n-1$, $(i = 0)$

giving a total of $(n-1)(n-2)/2$ different magnitudes. The same count exists for magnitudes $2^j - 2^i$; consequently, there exists a total of

$$N(2) = (n-1)(n-2) + 1 \tag{35}$$

weight-2 error magnitudes, including $(n-1)(n-2)$ magnitudes with the unqualified property M and one error magnitude $|E| = 2^{n-1} + 2^{n-2}$ with the restricted property M. Each error magnitude $|E|$ has a corresponding symmetrical pair of values $\pm |E|$, and, consequently, there are

$$2N(2) = 2(n-1)(n-2) + 2$$

distinct error numbers of weight 2 for the specified number length of n binary digits.

It remains to be determined how many and which weight-2 error magnitudes from the entire set of $(n-1)(n-2) + 1$ weight-2 magnitudes will remain undetected for $\alpha = 2^a - 1$. The undetectable magnitudes are

$$|E| = C(2^a-1), \text{ for } A/(2^a-1) > C > 0$$

and the weight-2 magnitudes are described by

$$|E| = 2^i(2^m \pm 1), \text{ with } A > |E| > 0$$

consequently, the solution of the equation

$$C(2^a-1) = 2^m \pm 1 \tag{36}$$

subject to the conditions

$$n \geq m \geq a \geq 2$$

for integer values of C in the range

$$A/(2^a-1) > C > 0$$

will yield all undetectable weight-2 error magnitudes of the form $2^m \pm 1$, that is, with the ± 1 digit in the right end position. The remaining undetectable weight-2 magnitudes will be of the form

$$|E| = 2^i C(2^a-1) = 2^i(2^m \pm 1), \text{ for } n-m > i > 0$$

To solve $C(2^a-1) = 2^m \pm 1$, observe first that C is an odd integer, since in $C\,2^a - C = 2^m \pm 1$ the right side is odd, while $C\,2^a$ is even. Furthermore, the only solution for

$$C(2^a-1) = 2^m + 1$$

exists when $a = 2$ and $m = 2j + 1$ (with $j \geq 1$), giving

$$C = (2^{2j+1} + 1)/3, \text{ for } a = 2,\ m = 2j + 1 \tag{37}$$

The remaining solution is required for

$$C(2^a-1) = 2^m - 1$$

where C is an odd integer in the range $A/(2^a-1) > C > 0$. An integer solution

$$C = 2^{(j-1)a} + \ldots + 2^a + 1 \tag{38}$$

exists for $m = ja$, that is, for $|E| = 2^{ja} - 1 < A$.

To count the undetectable weight-2 error magnitudes the error numbers are assumed to consist of

$$n = ka \tag{39}$$

binary digits and addition is modulo A, where

$$A = 2^n - 1 = 2^{ka} - 1 \tag{40}$$

The undetectable $|E|$ are now $|E| = 2^{ja} - 1$, for $k - 1 \geq j \geq 1$, and all multiples $2^i\,|E| < 2^{ka} - 1$. Counting the multiples, there are

a multiples for $j = k-1$; $(a - 1 \geq i \geq 0)$

$2a$ multiples for $j = k-2$; $(2a - 1 \geq i \geq 0)$

. . .

. . .

. . .

$(k-1)\,a$ multiples for $j = 1$; $(ka - a - 1 \geq i \geq 0)$

which give the total count of

$$\bar{N}(2) = a(1+2+\ldots+k-1) = (k-1)ka/2 \quad (41)$$

undetectable weight-2 forms $|E| = 2^{ja} - 1$. Since $k = n/a$ the count may be stated in terms of n and a

$$\bar{N}(2) = n(n-a)/2a \quad (42)$$

For the special case $a = 2$, there is undetectable $|E| = 2^{2j+1} + 1$ for $n > 2j + 1 > 1$, and this adds

$$\bar{N}(2)^* = (n-2)^2/4 \quad (43)$$

undetectable error magnitudes of weight 2 for the check modulus $\alpha = 3$.

Since there are $N(2) = (n-1)(n-2) + 1$ weight-2 error magnitudes, the fraction of undetected weight-2 error magnitudes for $a \geq 3$ is

$$\frac{\bar{N}(2)}{N(2)} = \frac{(n-a)}{2a(n-3) + 6a/n} < \frac{1}{2a} \quad (44)$$

For the case $a = 2$, the error count $\bar{N}(2)^*$ must be added, giving the fraction

$$\frac{\bar{N}(2) + \bar{N}(2)^*}{N(2)} = \frac{(n-1)(n-2)}{2[(n-1)(n-2) + 1]} \cong \frac{1}{2} \quad (45)$$

It may be concluded that the choice of $a \geq 3$ is quite effective in the detection of weight-2 error magnitudes. For example, given $n = 24$, the choice $a = 3$ leaves 16.6% weight-2 magnitudes undetected; $a = 4$ leaves 11.8%; and $a = 6$ leaves 7.1%. For $n = 25$ and $a = 5$, there are 9.0%. Since the values and forms of these magnitudes are known, further protection may be considered, if necessary.

It remains to be shown that the detectability of weight-1 and weight-2 error magnitudes corresponds to the detectability of a single use of one and two defective circuits, respectively. Given $A = B(2^a - 1)$, the two error magnitudes which may be caused by a single failure are

$$|E_j| = 2^j$$

$$|E_j| = B(2^a - 1) - 2^j$$

Since the only factors of 2^j are powers of two,

$$2^j \neq C(2^a - 1), \text{ for } a > 1, \text{ and } B > C > 0$$

All weight-1 error magnitudes will be detectable, and, therefore, all magnitudes $B(2^a - 1) - 2^j$ will also be detectable.

For two independent failures in positions j and i (with $j > i$), the value of the error number will be

$$E_2 \equiv E_j + E_i \text{ modulo } A; (n-1 \geq j > i \geq 0)$$

Both E_j and E_i will assume one of four values: $\pm 2^i$ and $\pm (A - 2^i)$. The sixteen different possibilities of modulo A addition give the eight values:

$$\pm(2^j + 2^i); \pm(2^j - 2^i); \pm[A - (2^j + 2^i)]; \pm[A - (2^j - 2^i)]$$

These values are reduced to four magnitude classes

$$|E_2| = 2^j \pm 2^i$$

$$|E_2| = A - (2^j \pm 2^i)$$

It has been shown that when $|E| = K$ is undetectable, then $|E| = A - K$ is also undetectable; consequently, the problem reduces to finding the undetectable error numbers of weight 2.

G. The Detection of Erasures with $\alpha = 2^a - 1$

The weights and the values of erasure numbers E' in a modulo $A = 2^n - 1$ system are of immediate interest. It is assumed that any position j of the n-digit result can be affected only once by the failed component, although the failure may affect several positions: the set $\{j\}$. The possible magnitudes of erasure numbers (for both up- and down-erasures) are

$$|E'| = \sum 2^j$$

$$|E'| = (2^n - 1) - \sum 2^j$$

Each affected position may be included or omitted in the summation; therefore, for m positions there are $2(2^m)$ possible values of $|E'|$. The undetectable erasures for $\alpha = 2^a - 1$ will be those satisfying

$$\sum 2^j = C(2^a - 1), B > C > 0 \quad (46)$$

and their digitwise complements.

Since the list of affected positions depends on the specific algorithms, the set of undetectable erasure magnitudes should be minimized by a careful design of the algorithms. For instance, excluding the number "all zeros" and using the value $A = 2^n - 1$ "all ones" as the only allowed representation of zero will permit the detection of all total down-erasures. The probability of a total up-erasure is reduced by avoiding completely serial algorithms and by disallowing the value zero (all ones) as an operand in ordinary multiplication and division algorithms.

Considerable protection against erasures may be expected from base-2^k operation (instead of base 2) in serial addition. In a parallel arithmetic unit recoding of multipliers and quotients allows base-4 algorithms (two bits at a time); a single parallel addition is not subject to an erasure. In both cases the algorithms are such that an erasure may not affect adjacent positions of the result. The effectiveness of specific distributions of susceptible positions is verified by solving the equation for undetectable erasures for every possible configuration of $\sum 2^j$.

H. Detectability of Controlled (Nonadjacent) Erasures

To determine the effectiveness of base-4 operation, it is necessary to count the percentages of erasures which will remain undetected for various values of α. It is assumed that base-4 operation will limit the list of affected positions in a result either to all even-numbered positions, or all odd-numbered positions. Consequently, an n-bit result will be subject to $2^{n/2} - 1$ possible values of E'. In the case $A = \alpha B$, the complements $A - E'$ will also be undetectable. An increase of two bits (from n to $n + 2$) in word length will add $2^{n/2}$ new possible values of E'. The solution of the equation

$$|E'| = C\alpha$$

for all $2^{n/2} - 1$ values of E' will give the percentage of undetectable erasures for a given word length n and for a given check modulus α. If complements of E' are partially or fully detectable, the percentage is appropriately reduced.

To determine the effectiveness of some convenient values of α, a computer program was written to count the undetectable erasure patterns E' (satisfying $|E'| = C\alpha$) for the check moduli

$$\alpha = 2^a - 1, \text{ with } 12 \geq a \geq 2$$

and word lengths $n = ka \leq 36$, for which $2^{ka} - 1 = (2^a - 1)\,B$ is satisfied and the product code is digit-wise complementable. The results are summarized in the Appendix, Section I. The tables list the total number of possible erasure patterns, the count of undetectable erasure patterns, and the percentage of undetectable errors for all word lengths $n = ka$. The incremental count and percentage of undetectable errors are also given for every increment of range. For an odd value of n, the worst case of $(n + 1)/2$ affected positions is chosen; for even values of n, there are $n/2$ affected positions.

For comparison the undetectable erasure patterns (base-4 operation) were also counted for distance 3 and distance 4 codes given by Peterson (Ref. 2, p. 239, table 13.1) which satisfy $\alpha B = 2^n - 1$. The results are given in the Appendix, Section II. It is interesting to observe that the effectiveness (percentage of undetectable erasure patterns) of these codes is similar to that of the $\alpha = 2^a - 1$ codes for the same word length and the same amount of redundancy.

In order to provide a list of all possible choices satisfying $2^n - 1 = \alpha B$, a list of the prime factors of $2^n - 1$ for $n \leq 100$ is given in the Appendix, Section III. The effectiveness of other values of α (expressed as the percentage of undetectable erasure patterns for base-4 operation) will be investigated and compared to the current results.

IV. ARITHMETICAL ALGORITHMS FOR PRODUCT-CODED BINARY NUMBERS

A. Introduction

The remaining problem (after the choice of a checking algorithm) is the design of a set of algorithms for an arithmetic unit which computes with product-coded numbers. The operands are products (αX, αY) instead of ordinary binary numbers (X, Y) and the result $S = \alpha X * \alpha Y$ should be the properly coded form $S = \alpha (X*Y)$ of the uncoded result $X*Y$.

It may be expected that because of these requirements there will be an increase in the cost of the arithmetic unit both in hardware and in the time required for the algorithm. It may be further expected that this cost will vary for different values of α, similar to the cost of the checking algorithm.

On the other hand, the probability of an undetected single defective circuit also depends on the hardware which is chosen to implement the algorithms. For instance, the logic design and length of the adder in the arithmetic unit determines which types of erasures and/or weight-1 errors will be caused by a single defective circuit within the adder.

The principal objective in the choice of algorithms is to hold both the additional cost and the probability of undetected defective circuits within reasonable bounds. The algorithms to be implemented are the usual repertoire of a general purpose digital computer: SUM, DIFFERENCE, LEFT SHIFT, RIGHT SHIFT, ROUNDOFF, PRODUCT, and QUOTIENT. The singularities "SUM OVERFLOW" and "QUOTIENT OVERFLOW" should be indicated. For all algorithms it is required that the output result should be the product-coded form of the result which would have been generated by an ordinary arithmetic unit. The n-digits-long binary operands are coded forms $S = (2^a - 1)X$.

B. The SUM and DIFFERENCE Algorithms

In the implementation of addition and subtraction the first choice is that of the representation of negative numbers; either sign-and-magnitude or complement forms may be used for this purpose. For the purpose of error detection the sign-and-magnitude forms are markedly inferior, since errors in the transmission and manipulation of the sign bit are nonnumerical and, therefore, not detectable by arithmetical checking.

In complement forms the sign is indicated by the value of the number; consequently, an error in the sign digit is arithmetically detectable. The complement of the value X in the range $M \geq X \geq 0$ with respect to A is defined as

$$c_A(X) = A - X, \text{ with } A > 2M \tag{47}$$

The two most convenient choices of A in conventional binary arithmetic with n-digits-long integers are

1. $A = 2^n$, called radix (two's) complement;
2. $A = 2^n - 1$, called digitwise (one's) complement.

These two choices are preferable because addition modulo A can be readily performed by either discarding the carry out of the leftmost position ($i = n-1$), or adding it in the rightmost position ($i = 0$).

For product-coded binary numbers there exists an additional requirement

$$c_A(\alpha X) = \alpha\, c_B(X) \tag{48}$$

where the product-coded numbers αX are complemented with respect to A and the uncoded numbers X are complemented with respect to B, that is

$$c_A(\alpha X) = A - \alpha X \tag{49}$$

$$c_B(X) = B - X \tag{50}$$

The requirement on complements is now expressed as $A - \alpha X = \alpha (B - X)$, which yields the condition

$$A = \alpha B \tag{51}$$

to be satisfied by the constants A and B for a given choice of α, which in our case is $\alpha = 2^a - 1$. Trying the two practical values of A, it is found that for $A = 2^n$ no integer solution exists for $B = 2^n/(2^a - 1)$ with $a \geq 2$. For the choice of $A = 2^n - 1$ there is

$$B = \frac{2^n - 1}{2^a - 1} \tag{52}$$

and integer solutions for B exist whenever $n = ka$ (integer $k \geq 1$), giving

$$B = 2^{(k-1)a} + 2^{(k-2)a} + \ldots + 2^a + 1 \qquad (53)$$

This value of B is not convenient for modulo B addition of uncoded numbers; however, uncoded numbers will not occur in the computer, and all additions will be performed modulo $A = 2^n - 1$, which is a relatively convenient choice.

The choice of B determines the range for the uncoded numbers X to be

$$B > 2M, \text{ or } B \geq 2M+1$$

which gives the greatest representable positive integer M as

$$M = (B-1)/2 = 2^{(k-1)\,a-1} + \ldots + 2^{2a-1} + 2^{a-1} \qquad (54)$$

The range of uncoded numbers X is, therefore,

$$M \geq X \geq -M$$

The advantage of this choice of M is that the coded numbers $Z = \alpha X$ form a one's-complement number system, in which the sign digit z_{n-1} indicates whether the number Z is in true form ($z_{n-1} = 0$) or in complement form ($z_{n-1} = 1$). The rules of a one's-complement arithmetic unit will apply in this case; however, it must be remembered that the numbers Z are actually products $(2^a-1)X$, and that X is the actual operand specified by the programmer.

The existence of two forms of the value "zero" – "all zeros" – corresponding to $(2^a-1)0$ and "all ones" corresponding to $(2^a-1)B = A = 2^n - 1$ can be used to advantage in checking by considering the "all zeros" form to be an error and, thus, detecting all total downerasures. It is also important to recollect that the choice of $B = A/(2^a-1)$ was also the most convenient choice in the evaluation of the effectiveness of check modulus $\alpha = 2^a - 1$.

The singularity SUM OVERFLOW must be indicated for coded numbers Z. In a fixed-point arithmetic unit this may be conveniently implemented by a comparison of the sign digits of the input operands (y_{n-1}, z_{n-1}) and of the sum (s_{n-1}). The case

$$y_{n-1} = z_{n-1} \neq s_{n-1} \qquad (55)$$

indicates that overflow has occurred.

In conventional floating-point addition the sum of two n-digit numbers is defined to be an $(n+1)$-digit number, which may have to be normalized by left or right shifts. Consequently, it is necessary to extend the range of the operands by one digit to the left and to form the sum of two $(n+1)$-digit numbers. Effectively, the value of A is changed from $2^n - 1$ to $2^{n+1} - 1$ (or from 2^n to 2^{n+1}).

For product-coded operands and $A = 2^{ka} - 1$ the next higher acceptable value A' should again satisfy

$$A' = (2^a-1)B'$$

The choice of $A' = 2^{ka+a} - 1$ gives the solution with the next higher integer value of B, which is

$$B' = 2^{ka} + 2^{(k-1)a} + \ldots + 2^a + 1 \qquad (56)$$

Given a complement form $Z = A - \alpha X$, to generate

$$A' - \alpha X = (A'-A) + (A-\alpha X) = Z + (A'-A)$$

the range extension is performed by adding $(A'-A)$ to the coded complement form Z. Observe that

$$A' - A = 2^{ka}(2^a-1) = 2^{ka} \sum_{0}^{a-1} 2^i \qquad (57)$$

and, consequently, the extension is performed by attaching a digits of value 1 to the left end of Z. If Z is a true form, a digits of value 0 are attached.

To reduce the length back to n digits, the $(n+a)$-digit number is shifted right until the $a+1$ leftmost digits are identical with the original sign digit; then a leftmost digits are dropped.

If correction of SUM OVERFLOW is the only objective of the extension, an actual extension by only one digit at the left is sufficient, since all a new digits will assume the same value after an addition (with or without OVERFLOW). The general extension/reduction rule remains in effect and is important for other algorithms.

C. The Arithmetic SHIFT Algorithms

The two arithmetic shifts of one position may be interpreted in terms of addition: the LEft Arithmetic Shift (LEAS) of Z as the addition Z + Z, and the RIght Arithmetic Shift (RIAS) as the addition Z + (−Z)/2. The

objective of the shift algorithms is to provide a fast method for the multiplication and division of the operand Z by the base, integer 2.

The LEAS algorithm should yield $W = \alpha(2X)$ for true forms ($Z = \alpha X$, $z_{ka-1} = 0$), and $W = A - \alpha(2X)$ for complement forms ($Z = A - \alpha X$, $z_{ka-1} = 1$). The shift is not arithmetical if OVERFLOW will occur, that is, if $z_{ka-1} \neq z_{ka-2}$ before the shift. The result W of the LEAS algorithm is described in terms of the digits of the operand Z as follows:

$$\text{LEAS:} \begin{cases} w_{i+1} = z_i, \; ka-2 \geq i \geq 0; \\ w_0 = z_{ka-1}; \\ \text{if } z_{ka-1} = z_{ka-2} \end{cases} \tag{58}$$

The "end-around" shift of z_{ka-1} is necessary to form the correct result for the complement forms as follows:

$$W = 2(A-\alpha X) - 2^{ka}+1 =$$

$$2A - (2^{ka}-1) - 2\alpha X = A - \alpha 2X$$

The RIAS algorithm should yield as its result V (given the operand Z), $V = \alpha(X/2)$ for true forms ($Z = \alpha X$), and $V = A - \alpha(X/2)$ for complement forms ($Z = A - \alpha X$). Observe that V may be exactly represented only for even values of X, since $\alpha X/2$ will only be an integer if X is even. (α is always odd.) If X is odd, ROUNDOFF (to the closest even X) must precede or be combined with the shift.

In true forms ($Z = \alpha X$), X will be even when Z is even, that is, when $z_{ka-1} = z_0 = 0$ is satisfied. In complement forms ($Z = A - \alpha X$), X will be even when Z is odd, that is, when $z_{ka-1} = z_0 = 1$ is satisfied. This is true because $A = 2^{ka} - 1$ is odd, and then $\alpha X = A - Z$ is even for odd values of Z. The result V of the RIAS algorithm is described in terms of the digits of operand Z as follows:

$$\text{RIAS:} \begin{cases} v_{i-1} = z_i, \; ka - 1 \geq i \geq 1; \\ v_{ka-1} = z_0; \\ \text{if } z_{ka-1} = z_0. \end{cases} \tag{59}$$

The "end-around" shift of z_0 again is required because of complement forms

$$V = (A-\alpha X-z_0)/2 + z_0\, 2^{ka-1} =$$

$$A/2 - \alpha X/2 - 1/2 + 2^{ka-1} = A - \alpha X/2$$

When the condition $z_{ka-1} = z_0$ is not satisfied, the ROUNDOFF algorithm (discussed below) must be applied to Z in order to satisfy the condition. Roundoff by truncation (discarding z_0) cannot be used, since the truncation will not yield a product-coded number.

Shifts over several positions may be considered as a sequence of single shifts. They may be executed if the specified conditions are satisfied for every single shift in the sequence.

D. The $(2^a - 1)$ Z Algorithm

Multiplication by the constant $2^a - 1$ is necessary in the implementation of the QUOTIENT algorithm. The constant $2^a - 1$ possesses properties which make the encoding of Z into $(2^a - 1)Z$ relatively fast and simple in a binary arithmetic unit.

Given a ka digit operand Z in true ($Z = \alpha X$) or in complement ($Z = A - \alpha X$) form, initially the range is extended by prefixing a 0-digits or a 1-digits at the left end of Z to get $(k+1)a$ digit operand $Z' = \alpha X$ or $Z' = A' - \alpha X$. The expansion of range has been discussed in connection with the SUM algorithm. Now Z' is shifted a positions to the left according to the LEAS algorithm, generating $2^a\alpha X$ or $A' - 2^a\alpha X$. The operand Z' is also complemented, generating $A' - \alpha X$ or αX. One addition modulo $A' = 2^{(k+1)a} - 1$ will yield

$$W = 2^a\alpha X + (A' - \alpha X) \equiv \alpha X(2^a - 1) \text{ modulo } A'; \text{ or}$$

$$W = (A' - 2^a\alpha X) + \alpha X \equiv A' - \alpha X(2^a - 1) \text{ modulo } A'$$

An "end-around" carry will occur if and only if Z is in true form and may be inserted in advance to speed up the addition.

The result will be $(k+1)a$ digits long; consequently, provisions must be made to store the extended number as well as to add modulo $A' = 2^{(k+1)a} - 1$. A circuit arrangement for serial multiplication by $2^a - 1$ is shown in Fig. 5; in this arrangement $(k+1)a$ steps of addition are required. The adder may add a digits at a time; then only $k+1$ steps are needed. Finally, a single step would be sufficient in a $(k+1)a$-digit, parallel adder.

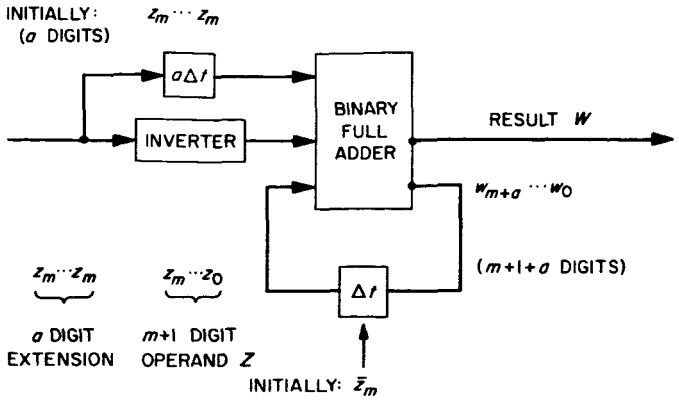

Fig. 5. Serial circuit for the $(2^a-1)Z$ algorithm with $m = ka-1$

E. The $W/(2^a - 1)$ Algorithm

Division by $2^a - 1$ is required in the PRODUCT algorithm, and is also very useful in the ROUNDOFF algorithm. The method for division by $2^a - 1$ has been discussed in the investigation of the checking algorithm; however, only the remainder was of interest there, and the quotient was actually never generated. The same considerations which led to the minimal cost checker can be applied to actual division by $2^a - 1$, in which the quotient is generated, and the remainder is known to be zero, since $Z = \alpha X$ or $Z = A - \alpha X = \alpha B - \alpha X = \alpha(B-X)$ will hold for every product-coded number in the computer.

When the remainder is known to be zero, division is an inverse operation to multiplication, which was described in the preceding section. Given the $(k+1)a$-digit forms W, the forms Z must be recovered:

1. true form: $W = \alpha X(2^a-1)$, recover $Z = \alpha X$;
2. complement: $W = A' - \alpha X(2^a-1)$, recover $Z = A - \alpha X$.

The algorithm may be generally stated as follows:

$$\bar{Z} = W + \overline{2^a Z} \text{ modulo } A' \qquad (60)$$

since $W = 2^aZ + \bar{Z}$ modulo A'; that is, the digits of $\bar{Z}$ may be recovered by adding $\overline{2^aZ}$ to the operand W, and accounting for the "end around" carry. The "end around" carry must be determined and employed in advance. The entire division process is possible because the rightmost a digits of 2^aZ are known to be the same as the sign digit of W (and of 2^aZ). The "end around" carry c_0 also depends on the sign digit (w_{ka-1+a}) of W. When W is a complement form, $c_0 = 1$. The carry occurs for W in complement form because the result $\bar{Z}$ is true and $\overline{2^aZ}$ is true; while for W in true form, $\overline{2^aZ}$ is complement, $\bar{Z}$ is complement, and no "end around" carry occurs.

When the "end around" carry and the a rightmost digits of $\overline{2^aZ}$ are known, the a rightmost digits of $\bar{Z}$ can be computed; they, in turn, are the next a digits of $\overline{2^aZ}$, etc. The digits of $\bar{Z}$ may be generated either serially, or a digits at a time, beginning with the rightmost digit(s). Designating $2^aZ = Y$, the algorithm is

$$\begin{aligned} c_0 &= w_{ka-1+a} \\ \bar{y}_j &= \bar{w}_{ka-1+a}, \text{ for } a-1 \geq j \geq 0 \\ \bar{y}_j &= \bar{z}_{j-a}, \text{ for } ka-1+a \geq j \geq a \qquad (61) \\ \bar{z}_j &= \bar{y}_j + w_j + c_j - 2c_{j+1}, \text{ for } ka-1+a \geq j \geq 0 \end{aligned}$$

c_{ka+a} is discarded.

The digits $\bar{z}_j$ for $ka-1 + a \geq j \geq ka$ will all be identical with $\bar{z}_{ka-1}$ and may be discarded, thus contracting $\bar{Z}$ to the length of ka digits.

A circuit for serial division by 2^a-1 is shown in Fig. 6; it requires ka steps of addition. A parallel adder for a digits at a time will permit completion in k steps. The circuit employs the same hardware as the multiplication circuit of Fig. 5.

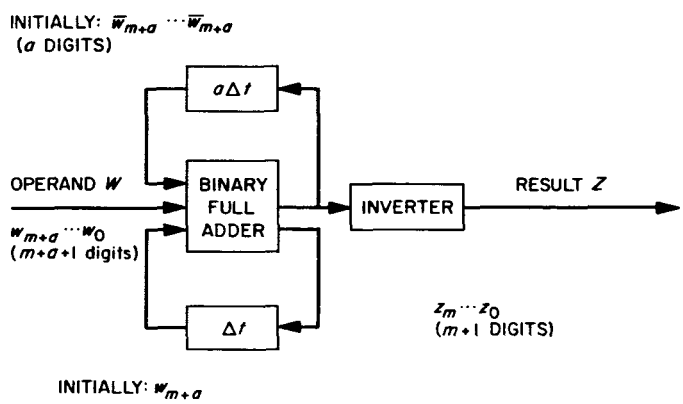

Fig. 6. Serial circuit for the $W/(2^a-1)$ algorithm with $m = ka-1$

The important observation here is that division has been replaced by modulo A' addition in which the "end

around" carry c_0 is known in advance to be equal to the sign digit w_{m+a} of the operand W, and is stored in the carry delay Δt. The rightmost a digits of $\overline{2^aZ}$ are also known. They are all equal to $\overline{w}_{m+a}$ and initially occupy the a-digits-long delay line or shift register $a\,\Delta t$. The algorithm for division by 2^a-1 is recursive; therefore, only a digits of the result Z can be generated at once. To get a digits at once, the single-digit adder of Fig. 6 is replaced by an a-digit, parallel adder, and the inputs are all a digits from the delay line $a\,\Delta t$ and a digits of the operand W.

F. The ROUNDOFF Algorithm

The ROUNDOFF algorithm is employed to reduce the length of a digital number by discarding one or more digits at the least significant end of the number and adjusting the value of the remaining digital number according to a specified roundoff rule. The simplest application of this algorithm occurs in the RIAS algorithm. The rightmost digit of a product-coded number may not be dropped, as for uncoded numbers, since removal of a digit may generate an unacceptable result. The RIAS algorithm requires the condition $z_{ka-1} = z_0$ to be satisfied. This condition assures that X is an even integer (X_e) in the coded forms $Z = \alpha X_e$ or $Z = A - \alpha X_e$. If the condition is not satisfied, X is an odd integer (X_0) and the rounded form $\alpha X_e = \alpha(X_0 \pm 1)$ must be generated before the shift. The rounded form Z' will have the coded values

$Z' = \alpha X_0 + \alpha$, or $Z' = A - \alpha X_0 - \alpha$, for up-rounding;

$Z' = \alpha X_0 - \alpha$, or $Z' = A - \alpha X_0 + \alpha$, for down-rounding.

The coded form ($\pm\,\alpha$) of $\pm\,1$ is added to Z to generate Z'; to avoid bias, rounding may be performed in both directions with equal probability. One convenient rounding rule for the case $z_{ka-1} = 0$ and $z_0 = 1$ is to add $\alpha = 2^a - 1$ by dropping z_0 and adding 2^a. In any case it is important to avoid overflow which may be caused by up-rounding a number of maximum allowed magnitude $|Z| = A - \alpha$.

The second and more general problem of roundoff is the reduction of a double-length result (for instance, a product) to the standard length. A simple truncation of the product-coded number will not yield a product-coded number of specified length. The coded number must be rounded by the addition or subtraction of an integral multiple $K(2^a-1)$ of the check modulus 2^a-1. The value of K is the value which is added or subtracted during the roundoff of an uncoded number.

Roundoff in conventional binary arithmetic is relatively simple: given a $2n$ digit number $X = 2^n F + G$, where F and G are n-digit numbers, so that G is in the range $2^n > G > 0$, the only two alternatives are

(a) form $X - G = 2^n F$, (rounding down);

(b) form $X + (2^n - G) = 2^n(F+1)$, (rounding up).

The first alternative is implemented by discarding G and the second is implemented by discarding G and adding 1 in the least significant position of F. The choice between (a) and (b) may depend on the value of G or on the value of the rightmost digit of F; or it may be always (a), which is the case of simple truncation.

Product-coded binary numbers αX should be rounded in exactly the same fashion. The roundoff procedure consists of the following steps

1. Compute the value of G by applying the $W/(2^a-1)$ algorithm to αX;
2. choose the rounding method (up or down);
3. compute $\alpha X - \alpha G$ for rounding down, or
4. compute $\alpha X + \alpha(2^n - G)$ for rounding up.

After the third step the rightmost n digits of the result are all 0's or all 1's (for true and complement forms of αX, respectively), and the result may be contracted to single length by discarding these digits. The detailed implementation of the ROUNDOFF algorithm is to be considered in further study of the arithmetical logic.

G. The PRODUCT Algorithm

The principal observation in the PRODUCT algorithm is that the operands are product-coded numbers, and the result of a conventional multiplication $(\alpha X)(\alpha Y) = \alpha^2 XY$ will not be the product-coded form αXY. The form $(2^a-1)XY$ is readily obtained by applying the $W/(2^a-1)$ algorithm to the intermediate result $(2^a-1)^2XY$. The PRODUCT algorithm is then concluded by the ROUNDOFF algorithm.

Given the ka digit operands $(2^a-1)X$ and $(2^a-1)Y$, the complete PRODUCT algorithm consists of the following steps

1. Extend the operand range to $2ka$ digits;
2. Perform a conventional one's-complement multiplication to generate the $2ka$-digit result $(2^a-1)P^* = (2^a-1)^2Z$;

3. Divide $(2^a-1)P^*$ by 2^a-1 using the $W/(2^a-1)$ algorithm to generate the $(2k-1)a$-digit product $P^* = (2^a-1)Z$;
4. Apply the ROUNDOFF algorithm to contract P^* to the ka-digit (single length) result $P = (2^a-1)F$.

The value of F should be the specified value of the single-length product of uncoded binary numbers X and Y, expressed in the system in which negative numbers are represented by complements with respect to $B = (2^{ka}-1)/(2^a-1)$.

Several problems remain to be resolved in the detailed logic design of the arithmetic unit. They include the application to fractional ranges, the selection of the most suitable one's-complement multiplication scheme and the efficient implementation of roundoff. Another problem of interest is the existence of undetectable single errors when the uncoded operands are multiples of the check modulus, that is, when $X = k_1(2^a-1)$ and $Y = k_2(2^a-1)$ is encountered. A further study of these problems will be presented in the detailed description of the arithmetic unit.

H. The QUOTIENT Algorithm

An ordinary binary division of dividend X by divisor Y yields a quotient Q and a remainder R such that $X = QY + R$ is satisfied, with $R < Y$. Dividing αX by αY gives $\alpha X = \alpha YQ + \alpha R$, and the quotient Q is generated in uncoded form; consequently, it is not protected against errors.

To generate a coded quotient αQ, start with the dividend $\alpha^2 X$ and in this case obtain

$$\alpha^2 X = (\alpha Q)(\alpha Y) + \alpha^2 R$$

The QUOTIENT algorithm for product-coded operands is composed of the following steps

1. Apply the $(2^a-1)Z$ algorithm to the dividend $Z = (2^a-1)X$;
2. Perform binary one's-complement division to get the ka-digit quotient $(2^a-1)Q$;
3. The coded remainder $(2^a-1)R$ is obtained from $(2^a-1)^2R$ by means of the $W/(2^a-1)$ algorithm.

It is interesting to observe that the intermediate result $(2^a-1)^2Z$ of the PRODUCT algorithm may be directly applied as the dividend $\alpha^2 X$ in the QUOTIENT algorithm.

The detailed design of the arithmetic unit requires the choice of a suitable one's-complement division method and the choice of a range for the coded operands. A method of indicating singularity QUOTIENT OVERFLOW (illegal division) is also required.

REFERENCES

1. G. W. Reitwiesner, "Binary Arithmetic", *Advances in Computers*, edited by F. L. Alt, Vol. 1, section 8, pp. 244-260, Academic Press Inc., New York, 1960.
2. Peterson, W. W., *Error Correcting Codes*, The Massachusetts Institute of Technology Press and John Wiley & Sons, Inc., New York, pp. 236-244, 1961.
3. Peterson, W. W., "On Checking an Adder", *IBM Journal of Research and Development*, Vol. 2, pp. 166-168, April, 1958.
4. Garner, H. L., "Generalized Parity Checking", *IRE Transactions on Electronic Computers*, Vol. EC-7, No. 3, pp. 207-213, September 1958.
5. Bloch, R. M., R. V. D. Campbell, M. Ellis, "The Logical Design of the Raytheon Computer", *Mathematical Tables and Other Aids to Computation*, Vol. 3, pp. 286-295; 317-323, October, 1948.
6. Bloch, R. M., "Diagnostic Information Monitoring System," U.S. Patent No. 2,634,052, application: April 27, 1949, patented: April 7, 1953, U.S. Department of Commerce, Washington, D.C.
7. Diamond, J. M., "Checking Codes for Digital Computers", *Proceedings of the IRE*, Vol. 43, pp. 487-488, April 1955.
8. Brown, D. T., "Error Detecting and Correcting Codes for Arithmetic Operations", *IRE Transactions on Electronic Computers*, Vol. EC-9, No. 3, pp. 333-337, September 1960.
9. Henderson, D. S., "Residue Class Error Checking Codes", preprints of papers presented at the 16th National Meeting of the Association for Computing Machinery, Los Angeles, September 5-8, 1961.
10. Kraitchik, M., *Recherches sur la Theorie des Nombres*, Gauthier-Villars, Paris, 1929, Vol. 2, pp. 84-88.

ACKNOWLEDGMENT

The author wishes to acknowledge many stimulating discussions with J. J. Wedel and A. D. Weeks. The assistance of A. D. Weeks in verification of the arithmetical algorithms by simulation and in the preparation of the tables in Appendix Sections I and II is gratefully acknowledged. The programs for the counting of erasure errors were written by E. L. Haskell, and the hardware for simulation of an independent checker was constructed by R. O. Davies.

APPENDIX

I. EFFECTIVENESS OF BASE-4 OPERATION FOR $\alpha = 2^a - 1$

Table A-1. $a = 2, \alpha = 3$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
2	1	1	0	0	0	0
4	3	2	0	0	0	0
6	7	4	1	14.3	1	25.0
8	15	8	4	26.7	3	37.5
10	31	16	10	32.3	6	37.5
12	63	32	21	33.3	11	34.4
14	127	64	42	33.1	21	32.8
16	255	128	84	32.9	42	32.8
18	511	256	169	33.1	85	33.2
20	1023	512	340	33.2	171	33.4
22	2047	1024	682	33.3	342	33.4
24	4095	2048	1365	33.3	683	33.3
26	8191	4096	2730	33.3	1365	33.3
28	16383	8192	5460	33.3	2730	33.3
30	32767	16384	10921	33.3	5461	33.3
32	65535	32768	21844	33.3	10923	33.3
34	131071	65536	43690	33.3	21846	33.3
36	262143	131072	87381	33.3	43691	33.3

Table A-2. $a = 3, \alpha = 7$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
3	3	3	0	0	0	0
6	7	4	1	14.3	1	25.0
9	31	24	4	12.9	3	12.5
12	63	32	9	14.3	5	15.6
15	255	192	36	14.1	27	14.1
18	511	256	73	14.3	37	14.5
21	2047	1536	292	14.3	219	14.3
24	4095	2048	585	14.3	293	14.3
27	16383	12288	2340	14.3	1755	14.3
30	32767	16384	4681	14.3	2341	14.3
33	131071	98304	18724	14.3	14043	14.3
36	262143	131072	87381	14.3	43691	14.3

Table A-3. $a = 4, \alpha = 15$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
4	3	3	0	0	0	0
8	15	12	0	0	0	0
12	63	48	1	1.59	1	2.08
16	255	192	16	6.27	15	7.81
20	1023	768	100	9.78	84	10.90
24	4095	3072	401	9.79	301	9.80
28	16383	12288	1316	8.03	915	7.45
32	65535	49152	4368	6.67	3052	6.21
36	262143	196608	16705	6.37	12337	6.27

Table A-4. $a = 5, \alpha = 31$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
5	7	7	0	0	0	0
10	31	24	1	3.23	1	4.17
15	255	224	8	3.14	7	3.13
20	1023	768	33	3.23	25	3.26
25	8191	7168	264	3.22	231	3.22
30	32767	24576	1057	3.23	793	3.23
35	262143	229376	8456	3.23	7399	3.23

Table A-5. $a = 6, \alpha = 63$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
6	7	7	0	0	0	0
12	63	56	0	0	0	0
18	511	448	1	0.20	1	0.22
24	4095	3584	64	1.56	63	1.76
30	32767	28672	1000	3.05	936	3.26
36	262143	229376	8001	3.05	7001	3.05

Table A-6. $a = 7, \alpha = 127$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
7	15	15	0	0	0	0
14	127	112	1	0.79	1	0.89
21	2047	1920	16	0.78	15	0.78
28	16383	14336	129	0.79	113	0.79
35	262143	245760	2064	0.79	1935	0.79

Table A-7. $a = 8, \alpha = 255$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
8	15	15	0	0	0	0
16	255	240	0	0	0	0
24	4095	3840	1	0.24	1	0.26
32	65535	61440	256	0.39	255	0.42

Table A-8. $a = 9, \alpha = 511$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
9	31	31	0	0	0	0
18	511	480	1	0.20	1	0.20
27	16383	15872	32	0.20	31	0.20
36	262143	245760	513	0.20	481	0.20

Table A-9. $a = 10, \alpha = 1023$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
10	31	31	0	0	0	0
20	1023	992	0	0	0	0
30	32767	31744	1		1	

Table A-10. $a = 11, \alpha = 2047$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
11	63	63	0	0	0	0
22	2047	1984	1	0.05	1	0.05
33	131071	129024	64	0.05	63	0.05

Table A-11. $a = 12, \alpha = 4095$

n	Count of possible erasure numbers		Total undetectable erasure numbers		Increment of undetectable erasure numbers	
	Total	Increment	Count	%	Count	%
12	63	63	0	0	0	0
24	4095	4032	0	0	0	0
36	262143	258048	1		1	

II. EFFECTIVENESS OF BASE-4 OPERATION FOR FIXED DISTANCE CODES

Table A-12. Distance 3 codes

n	α	Count of possible erasure numbers	Undetectable erasure numbers	
			Count	%
20	55	1023	23	2.25
20	75	1023	20	1.96
21	49	2047	46	2.25
22	69	2047	33	1.61
23	47	4095	90	2.20
28	87	16383	196	1.96
30	77	32767	421	1.28
35	71	262143	3700	1.41
36	95	262143	2881	1.10

Table A-13. Distance 4 codes

n	α	Count of possible erasure numbers	Undetectable erasure numbers	
			Count	%
11	89	63	0	0
12	105	63	1	1.59
22	267	2047	0	0
24	357	4095	33	0.81
36	555	262143	469	0.18

III. FACTORIZATION OF 2^n-1 INTO PRIMES

Table A-14 indicates all values of α which satisfy the condition $2^n-1 = \alpha B$. Any prime factor or any product of prime factors will serve as α for a digitwise complementable product code with $A = 2^n-1$ (n-bit words). The source of Table A-14 is Ref. 10.

Table A-14. Prime factors of 2^n-1

n	Prime factors
2	3
3	7
4	3 x 5
5	31
6	3 x 3 x 7
7	127
8	3 x 5 x 17
9	7 x 73
10	3 x 11 x 31
11	23 x 89
12	3 x 3 x 5 x 7 x 13
13	8191
14	3 x 43 x 127
15	7 x 31 x 151
16	3 x 5 x 17 x 257
17	131071
18	3 x 3 x 3 x 7 x 19 x 73
19	524,287
20	3 x 5 x 5 x 11 x 31 x 41
21	7 x 7 x 127 x 337
22	3 x 23 x 89 x 683
23	47 x 178,481
24	3 x 3 x 5 x 7 x 13 x 17 x 241
25	31 x 601 x 1,801
26	3 x 2731 x 8,191
27	7 x 73 x 262,657
28	3 x 5 x 29 x 43 x 113 x 127
29	233 x 1,103 x 2,089
30	3 x 3 x 7 x 11 x 31 x 151 x 331
31	2,147,483,647
32	3 x 5 x 17 x 257 x 65,537
33	7 x 23 x 89 x 599,479
34	3 x 43,691 x 131,071
35	31 x 71 x 127 x 122,921
36	3 x 3 x 3 x 5 x 7 x 13 x 19 x 37 x 73 x 109
37	223 x 616,318,177
38	3 x 174,763 x 524,287
39	7 x 79 x 8,191 x 121,369
40	3 x 5 x 5 x 11 x 17 x 31 x 41 x 61,681
41	13,367 x 164,511,353
42	3 x 3 x 7 x 7 x 43 x 127 x 337 x 5,419
43	431 x 9,719 x 2,099,863
44	3 x 5 x 23 x 89 x 397 x 683 x 2,113
45	7 x 31 x 73 x 151 x 631 x 23,311
46	3 x 47 x 178,481 x 2,796,203
47	2,351 x 4,513 x 13,264,529
48	3 x 3 x 5 x 7 x 13 x 17 x 97 x 241 x 257 x 673

Table A-14. Prime factors of $2^n - 1$ (Cont'd)

n	Prime factors
49	127 x 4,432,676,798,593
50	3 x 11 x 31 x 251 x 601 x 1,801 x 4,051
51	7 x 103 x 2,143 x 11,119 x 131,071
52	3 x 5 x 53 x 157 x 1,613 x 2,731 x 8,191
53	6,361 x 69,431 x 20,394,401
54	3 x 3 x 3 x 3 x 7 x 19 x 73 x 87,211 x 262,657
55	23 x 31 x 89 x 881 x 3,191 x 201,961
56	3 x 5 x 17 x 29 x 43 x 113 x 127 x 15,790,321
57	7 x 32,377 x 524,287 x 1,212,847
58	3 x 59 x 233 x 1,103 x 2,089 x 3,033,169
59	179,951 x 320,343 x 1,780,337
60	3 x 3 x 5 x 5 x 7 x 11 x 13 x 31 x 41 x 61 x 151 x 331 x 1,321
61	2,305,843,009,213,693,951
62	3 x 715,827,883 x 2,147,483,647
63	7 x 7 x 73 x 127 x 337 x 92,737 x 649,657
64	3 x 5 x 17 x 257 x 641 x 65,537 x 6,700,417
65	31 x 8,191 x 145,295,143,558,111
66	3 x 3 x 7 x 23 x 67 x 89 x 683 x 20,857 x 599,479
67	193,707,721 x 761,838,257,287
68	3 x 5 x 137 x 953 x 26,317 x 43,691 x 131,071
69	7 x 47 x 178,481 x 10,052,678,938,039
70	3 x 11 x 31 x 43 x 71 x 127 x 281 x 86,171 x 122,921
71	228,479 x 48,544,121 x 212,885,833
72	3 x 3 x 3 x 5 x 7 x 13 x 17 x 19 x 37 x 73 x 109 x 241 x 433 x 38,737
73	439 x 2,298,041 x 9,361,973,132,609
74	3 x 223 x 1,777 x 25,781,083 x 616,318,177
75	7 x 31 x 151 x 601 x 1,801 x 100,801 x 10,567,201
76	3 x 5 x 229 x 457 x 174,763 x 524,287 x 525,313
77	23 x 89 x 127 x 581,283,643,249,112,959
78	3 x 3 x 7 x 79 x 2,731 x 8,191 x 121,369 x 22,366,891
79	2,687 x 202,029,703 x 1,113,491,139,767
80	3 x 5 x 5 x 11 x 17 x 31 x 41 x 257 x 61,681 x 4,278,255,361
81	7 x 73 x 2,593 x 71,119 x 262,657 x 97,685,839
82	3 x 83 x 13,367 x 164,511,353 x 8,831,418,697
83	
84	3 x 3 x 5 x 7 x 7 x 13 x 29 x 43 x 113 x 127 x 337 x 1,429 x 5,419 x 14,449
85	31 x 131,071 x 9,520,972,806,333,758,431
86	3 x 431 x 9,719 x 2,099,863 x 2,932,031,007,403
87	7 x 233 x 1,107 x 2,089 x 4,177 x 9,857,737,155,463
88	3 x 5 x 17 x 23 x 89 x 353 x 397 x 683 x 2,113 x 2,931,542,417
89	618,970,019,642,690,137,449,562,111
90	3 x 3 x 3 x 7 x 11 x 19 x 31 x 73 x 151 x 331 x 631 x 23,311 x 18,837,001
91	127 x 911 x 8,191 x 112,901,153 x 23,140,471,537
92	3 x 5 x 47 x 277 x 1,013 x 1,657 x 30,269 x 178,481 x 2,796,203
93	7 x 2,147,483,647 x 658,812,288,653,553,079
94	3 x 283 x 2,351 x 4,513 x 13,264,529 x 165,768,537,521
95	31 x 191 x 524,287 x 12,761,021,422,289,693,921
96	3 x 3 x 5 x 7 x 13 x 17 x 97 x 193 x 241 x 257 x 673 x 65,537 x 22,253,377
97	
98	3 x 43 x 127 x 4,363,953,127,297 x 4,432,676,798,593
99	7 x 23 x 73 x 89 x 199 x 153,649 x 599,479 x 33,057,806,959
100	3 x 5 x 5 x 5 x 11 x 31 x 41 x 101 x 251 x 601 x 1,801 x 4,051 x 8,101 x 268,501